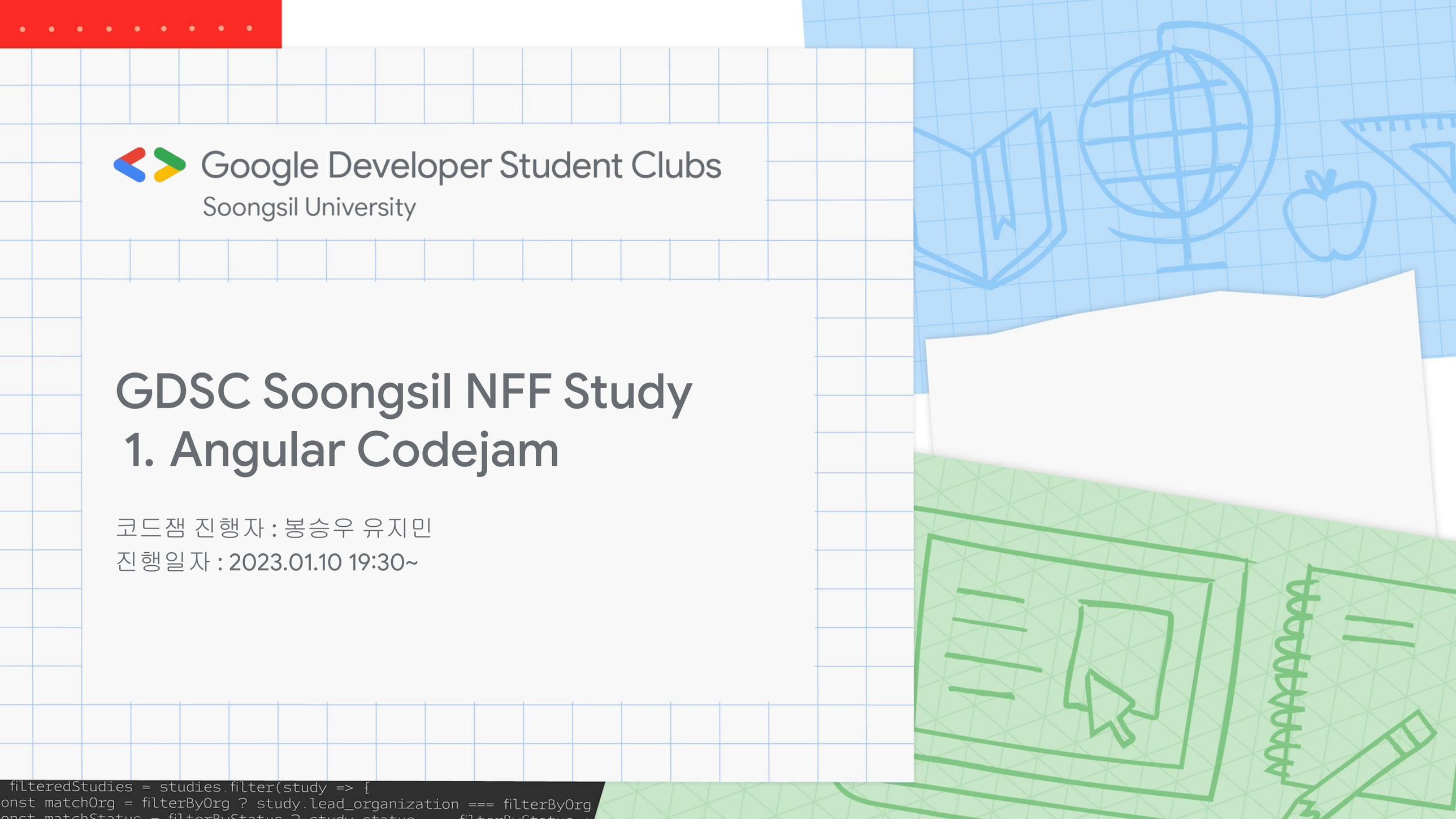
Google Developer Student Clubs
Soongsil University

GDSC Soongsil NFF Study
1. Angular Codejam

코드잼 진행자 : 봉승우 유지민
진행일자 : 2023.01.10 19:30~



What is Angular ?

- **Angular : SPA** 개발을 위한 구글 오픈소스, 자바스크립트의 프레임워크
  - 웹 애플리케이션 + 모바일 웹 + 네이티브 모바일 + 데스크탑 앱 개발 가능
  - 주력 언어 : **Typescript** (정적 타입 제공) 채택
- **AngularJS : 2012, ver1 (ver2**는 **Angular)**

- **AngularJS : Controller, $scope** 기반 개발 → **Angular : CBD(**컴포넌트 기반 개발**)**로 전환
  AngularJS : angular.module & jQlite → Angular : 향상된 모듈 시스템 **+ DOM** 제어 기능 제공 **+ API**의 단순화
- **Angular :** 선택적 데이터 바인딩 지원 **(one-way & two-way,** 양방향 데이터 바인딩을 빌트인으로 제공하지 않음**)**
- **Angular :** 디렉티브 **&** 서비스 **&** 의존성 주입의 간소화
  주력 언어 **: JS → TS :** 대규모 개발에 적합한 정적 타입 및 인터페이스 **,**제네릭 등 타입 체크 기능 제공
- **ECMAScript6**에서 도입된 모듈, 클래스 **+ ES7** 데코레이터 지원
- **Angular CLI** 제공

What is Angular ?

데이터 바인딩	데이터의 흐름	문법
인터폴레이션	컴포넌트 클래스 ⟹ 템플릿	{{ expression }}
프로퍼티 바인딩	컴포넌트 클래스 ⟹ 템플릿	[property]="expression"
어트리뷰트 바인딩	컴포넌트 클래스 ⟹ 템플릿	[attr.attribute-name]="expression"
클래스 바인딩	컴포넌트 클래스 ⟹ 템플릿	[class.class-name]="expression"
스타일 바인딩	컴포넌트 클래스 ⟹ 템플릿	[style.style-name]="expression"
이벤트 바인딩	컴포넌트 클래스 ⟸ 템플릿	(event)="statement"
양방향 데이터 바인딩	컴포넌트 클래스 ⟺ 템플릿	[(ngModel)]="property"

What is Angular ?

1. 개발 생산성 개선

- CBD (Component Based Development) : 개발 생산성 및 대규모 애플리케이션에 적합
  - ↔ AngularJS : Controller & $scope 중심
- Typescript 도입 : ES6(모듈, 클래스) + ES7(데코레이터) 지원
  - 도구 지원 (정적 타이핑을 지원하기에 높은 수준에 인텔리센스, 코드 어시스트, 타입체크, 리팩토링)
  - 명시적인 정적 타입 지정으로 코드 가독성 향상 및 예측 가능
  - 컴파일 단계에서 오류 포착 가능
  - 모듈, 클래스, 인터페이스 등 OOP 지원
  - Angular : TS, JS, Dart로도 작성 가능 → 공식 문서 : TS 기반
- 개발 도구의 통합 및 개발 환경 구축 자동화 : Angular CLI

What is Angular ?

2. 성능 향상

- **Digest Loop**로 인한 성능저하 문제 해결
  AngularJS : Digest Loop(Model의 변화를 **View**에 반영시키는 과정**)**에서 성능 저하
  - 양방향 데이터 바인딩 추가 시 **: watcher** 추가 필요 → **watcher**에 대해 **Digest Loop** 실행되기에 **watcher** 늘어날수록 성능 저하
  - 로딩 시간 **: 2.5**배 향상
  - 리렌더링 **: 4.2**배 향상

- **AoT** 컴파일
  Ahead of Time compilation : 사전 컴파일 방식
  - 구조 디렉티브를 브라우저가 실행 가능한 코드로 변환
    - 런타임이 아닌 사전에 컴파일해 실행 속도 향상
  - **JIT(Just in Time)**컴파일러가 필요 없어 프레임워크 크기를 **50%** 줄일 수 있음

- **Lazy Loading**
  SPA의 단점 극복 대안
  - 애플리케이션 실행 시점에 애플리케이션에서 사용되는 모든 모듈을 한꺼번에 로딩 **X**
    → 필요한 시점에 필요한 모듈만을 로딩하는 방식
    → 페이지 로딩 속도 향상 가능

- 코드 최적화
  - **Angular : Mobile First** 지향

Angular의 4대 요소

- 컴포넌트 : 사용자에게 보여지는 뷰 영역
  - **@Component** 라는 데코레이터로 정의된 클래스

- 디렉티브 : **Angular** 애플리케이션 내부 엘리먼트에 어떤 동작을 추가하는 클래스
  - **Components, Attribute directives, Structural directive**
  - **ex) NgClass, NgStyle, NgModel, ...**

- 서비스 : 뷰를 제외한 로직 영역
  - **@Injectable** 데코레이터가 붙은 클래스

- 모듈 : 앵귤러의 서비스 단위. 컴포넌트, 디렉티브, 서비스 등을 묶어 의존성 관리
  - **@NgModule** 데코레이터가 붙은 클래스

- **+)** 데코레이터 : 해당 파일에 대해 앵귤러가 정의한 기능을 부여하는 데 사용
  - **ex) @Component, @Injectable, @NgModule ...**

Angular Component

컴포넌트는 독립적이고 완결된 뷰를 생성하기 위하여 **“HTML, CSS,** 자바스크립트를 하나의 단위로 묶는 것”

W3C 표준인 웹 컴포넌트**(Web Component)**를 기반으로 함

1. 컴포넌트의 뷰를 생성할 수 있어야 하며**(HTML Template)**
2. 외부로부터의 간섭을 제어하기 위해 스코프**(scope)**를 분리하여 **DOM**을 캡슐화**(Encapsulation)**할 수 있어야 하며**(Shadow DOM)**
3. 외부에서 컴포넌트를 호출할 수 있어야 하고**(HTML import)**
4. 컴포넌트를 명시적으로 호출하기 위한 명칭**(alias)**을 선언하여 마치 네이티브 **HTML** 요소와 같이 사용할 수 있어야 한다 **(Custom Element).**

- **Angular**의 컴포넌트는 **Web Component**의 개념과는 다름

컴포넌트 기본 동작 구조

src/app/app.component.ts

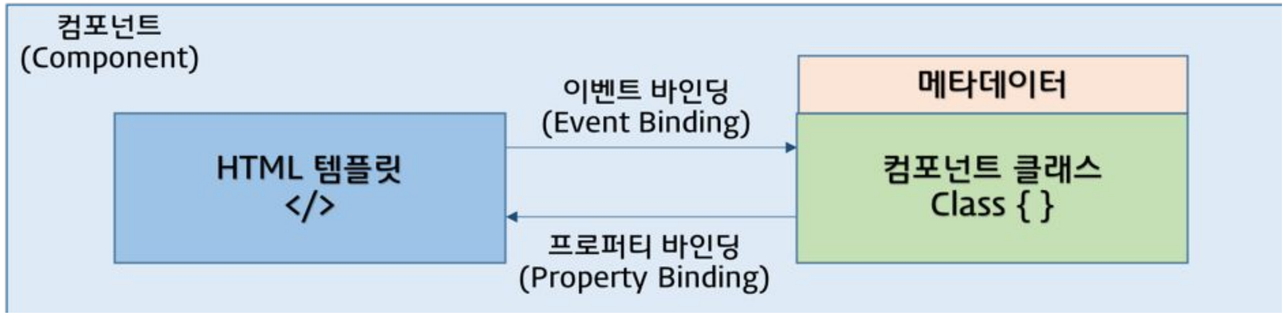
```ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app';
}
```

src/app/app.component.html

```html
<h1>
  Welcome to {{ title }}!
</h1>
```

Data Binding



컴포넌트 기본 동작 구조

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-hello',
5     template: `
6       <h2>안녕하세요 {{name}}</h2>
7       <input type="text" placeholder="이름을 입력하세요" #inputYourName>
8       <button (click)="setName(inputYourName.value)">등록</button>
9     `,
10    styles: [`…
39    `]
40  })
41  export class HelloComponent {
42    name: string;
43
44    setName(name: string) {
45      this.name = name;
46    }
47  }
```

③

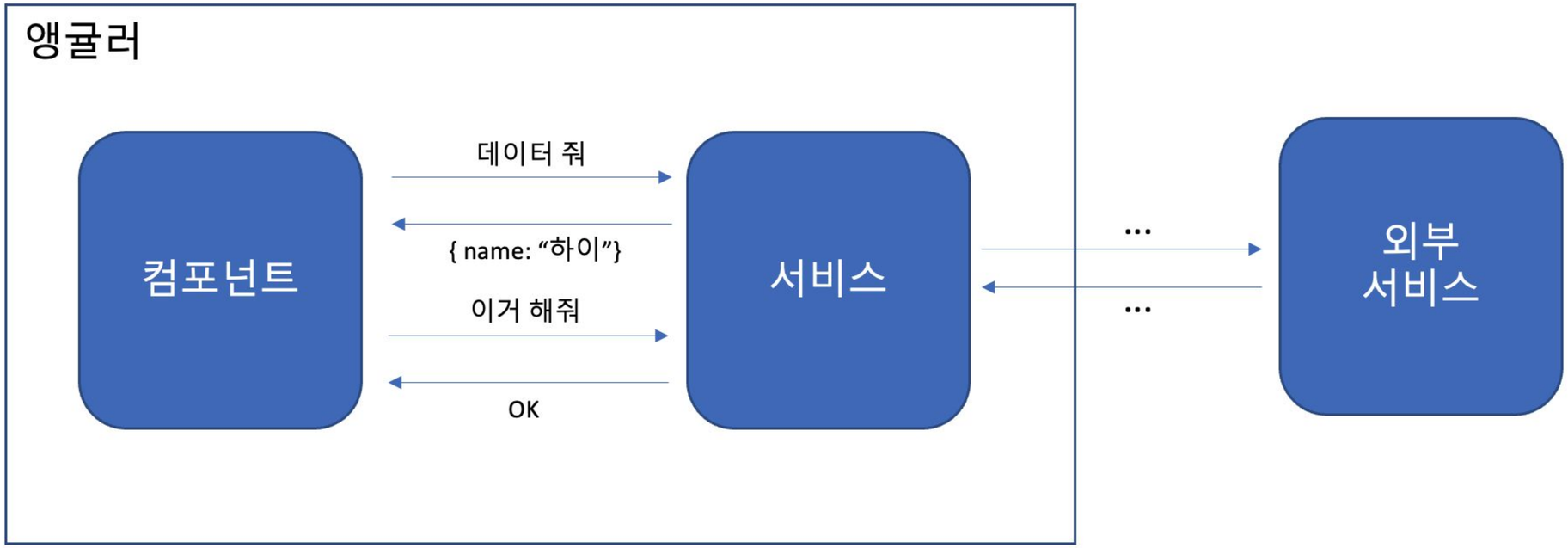
①

②

서비스

서비스 : 화면을 구성하는 뷰를 생성하고 관리 하는 것이 주된 역할
- 주 관심사 이외의 부가적인 기능 필요
- 컴포넌트의 재사용성 감소 & 코드 중복 증가 & 복잡도 상승

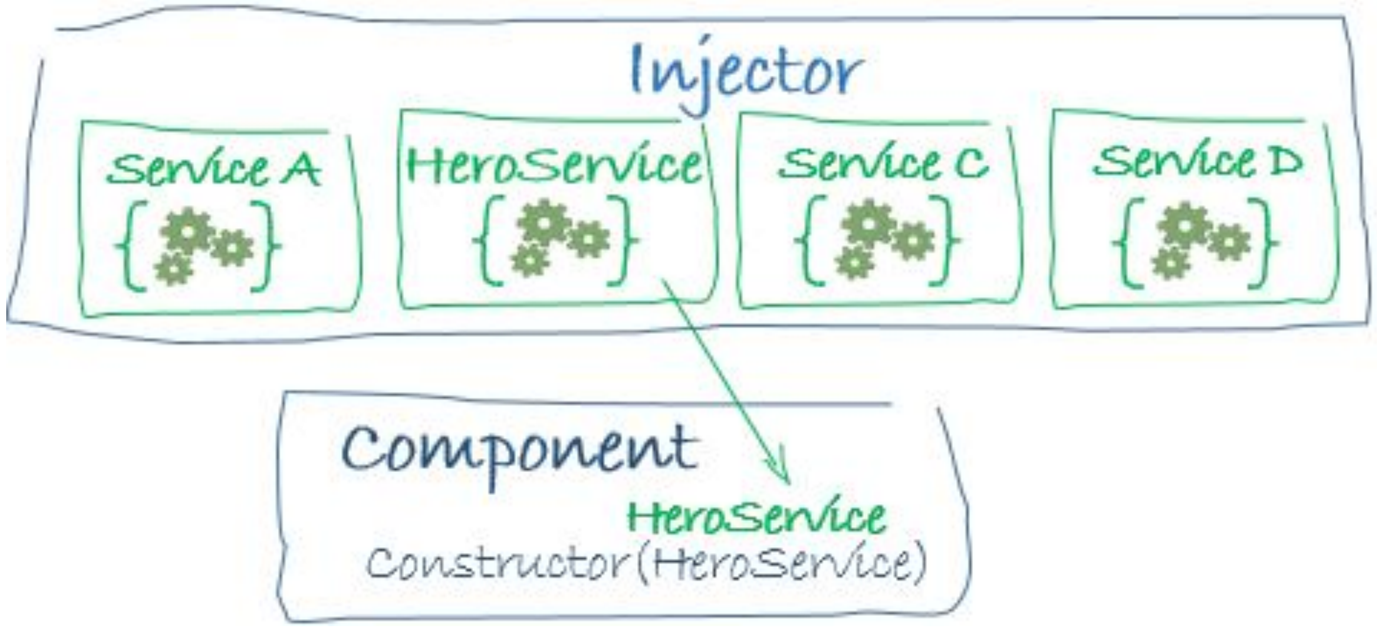
- 서비스를 통해 애플리케이션 전역의 관심사 분리

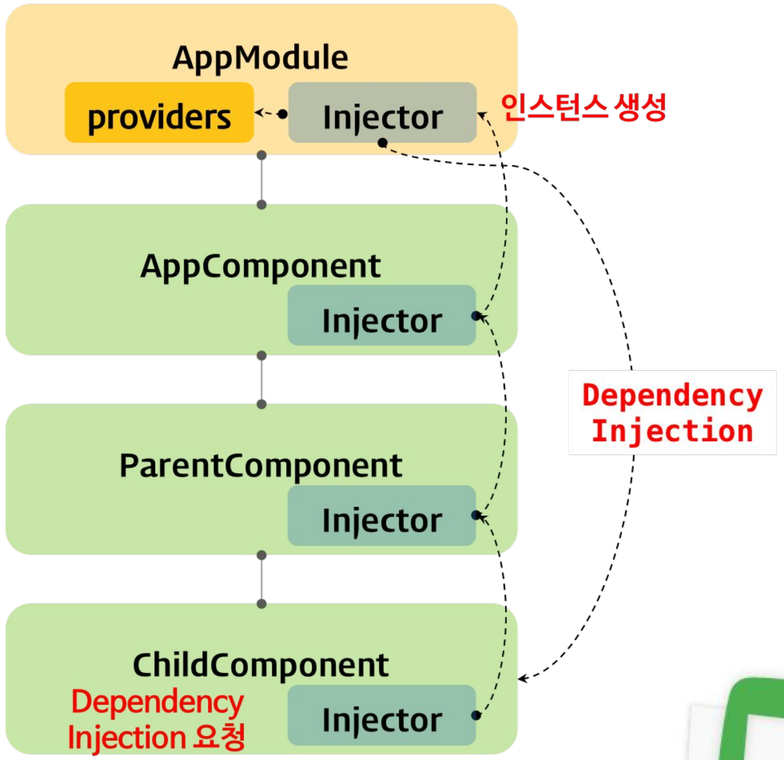


이미지 출처 : **itjustbong**님 벨로그 **(**여기 글 짱 좋아요**!!!)**

서비스 - 인젝터

의존성 주입 요청에 의해 주압되어야 할 인스턴스 O ? 인젝터에 요청
- 인젝터 ? 프로바이더 ?
- 인젝터 트리 ?





인젝터 트리

모듈

모듈 : 관련이 있는 구성요소(컴포넌트, 디렉티브, 파이프, 서비스 등)을 하나의 단위로 묶는 메커니즘
- 결합도를 낮추고 응집도를 높이는 것이 **Best!**

- 앵귤러로 만든 애플리케이션 : 최소 하나 이상의 모듈의 결합으로 이루어짐
  ⇨ 최상위 모듈이 존재한다는 의미 == 루트 모듈 **(AppModule)**

```
...
@NgModule({
  declarations: [...],
  imports: [...],
  providers: [...],
  bootstrap: [...]
})
 export class AppModule { }
```

디렉티브

디렉티브 : 애플리케이션 안에 있는 엘리먼트에 어떤 동작을 추가하는 클래스

```
<!--An extra Div element will be added-->
<div *ngIf="shouldShow">
    <div *ngFor="let order of orders">
        <li>Order:{{order.ordername}}</li>
    </div>
  </div>

<!--ng-container wont be added to the DOM-->
<ng-container *ngIf="shouldShow">
    <div *ngFor="let order of orders">
        <li>Order:{{order.ordername}}</li>
    </div>
</ng-container>
```

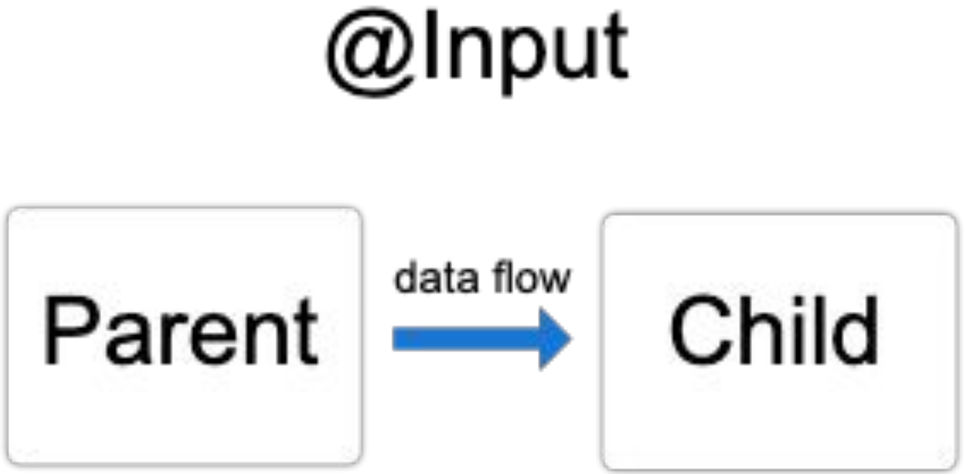
using *ngFor and *ngIf on same element

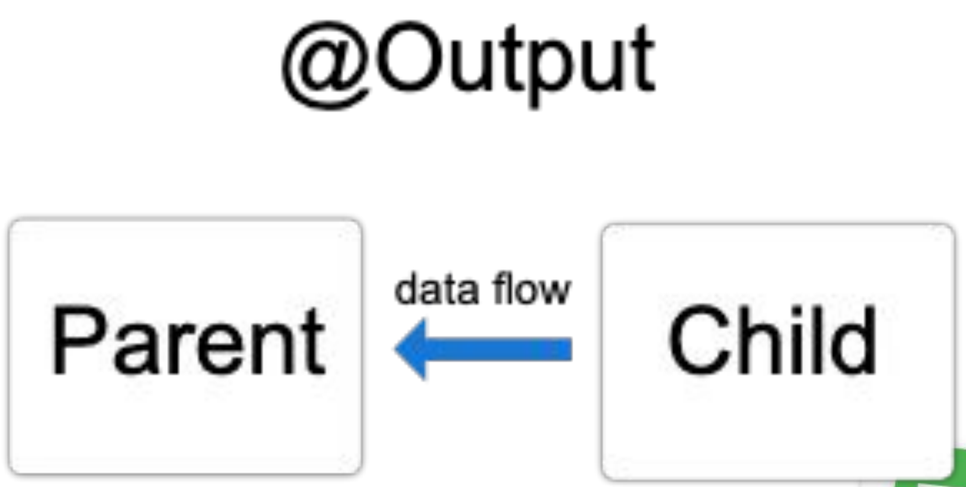
기본 구조 디렉티브	설명
NgIf	조건에 따라 템플릿의 일부를 DOM 트리에 추가하거나 DOM 트리에서 제거합니다.
NgFor	배열에 있는 항목마다 템플릿 일부를 반복합니다.
NgSwitch	조건에 맞는 화면을 DOM 트리에 추가합니다.

데코레이터

- 데코레이터 : 해당 파일에 대해 앵귤러가 정의한 기능을 부여하는 데 사용
  - **ex) @Component, @Injectable, @NgModule ...**

- **ex)** 부모 컴포넌트와 자식 컴포넌트가 데이터를 주고받는 패턴
  - **@Input, @Output** 데코레이터 사용





Angular CLI

- **Angular CLI** 지원 기능
  - **Angular** 프로젝트 생성
  - **Angular** 프로젝트에 컴포넌트, 디렉티브, 파이프, 서비스, 클래스, 인터페이스 등의 구성 요소 추가
  - **LiveReload**를 지원하는 내장 개발 서버를 사용한 **Angular** 프로젝트 실행
  - **Unit / E2E(end-to-end)** 테스트 환경 지원
  - 배포를 위한 **Angular** 프로젝트 패키징

`sudo npm install -g @angular/cli` : **Angular CLI** 설치 **(macOS)**

`ng –version` : **Angular** 버전 확인

`ng new <project-name>` : **Angular** 프로젝트 생성

`cd <project-name>`

`ng serve` : 프로젝트 실행

Angular CLI

```
my-app/
├── .git/
├── e2e/
├── node_modules/
├── src/
├── .editorconfig
├── .gitignore
├── angular.json
├── package-lock.json
├── package.json
├── README.md
├── tsconfig.json
└── tslint.json
```

Angular Style Guide에 따라 설계된 구조
- 기본 애플리케이션 구조
- 네이밍 룰
- 코딩 컨벤션

Angular CLI

생성 대상 구성요소	명령어	축약형
컴포넌트	ng generate component component-name	ng g c component-name
디렉티브	ng generate directive directive-name	ng g d directive-name
파이프	ng generate pipe pipe-name	ng g p pipe-name
서비스	ng generate service service-name	ng g s service-name
모듈	ng generate module module-name	ng g m module-name
가드	ng generate guard guard-name	ng g g guard-name
클래스	ng generate class class-name	ng g cl class-name
인터페이스	ng generate interface interface-name	ng g i interface-name
Enum	ng generate enum enum-name	ng g e enum-name

- 파일명이 암묵적으로 변경됨 **(kebab-case)**

ng generate component newComponent

ng generate component NewComponent

ng generate component new-component

모두 new-component.component.*로 저장됨

Angular CLI

- **templateUrl :** 외부 파일로 작성된 **HTML** 템플릿(컴포넌트의 뷰를 정의)의 경로
- **styleUrls :** 외부 파일로 작성된 **CSS** 파일 경로

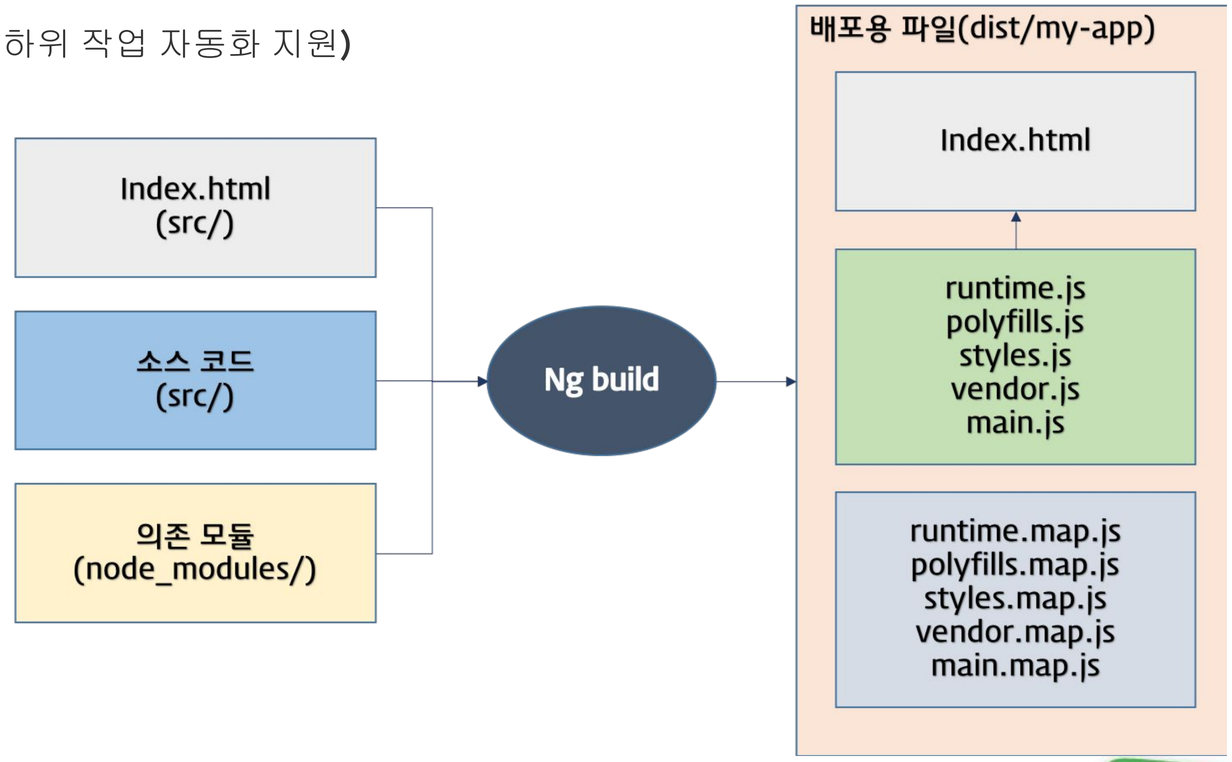
```
// src/app/home/home.component.ts
...
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
...
```

```
// src/app/home/home.component.ts
...
@Component({
  selector: 'app-home',
  template: `
    <p>home works!</p>
  `,
  styles: [`
    p { color: red; }
  `]
})
...
```

Angular CLI

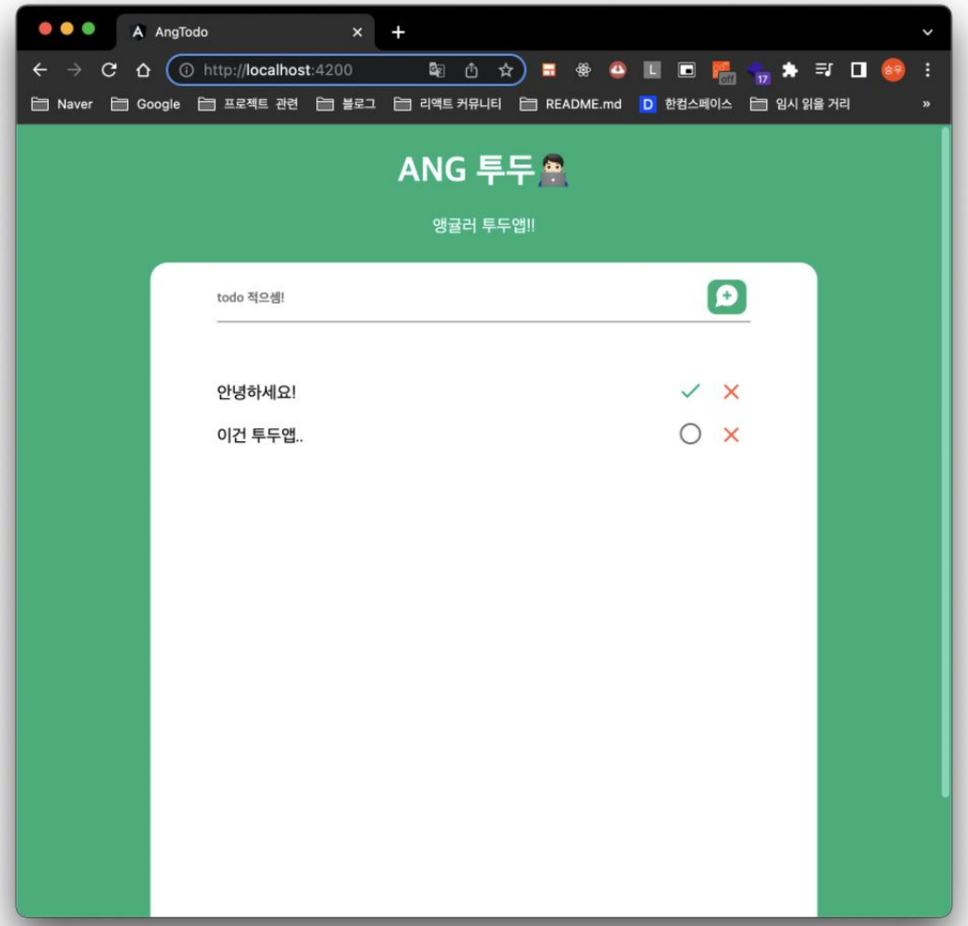
ng build

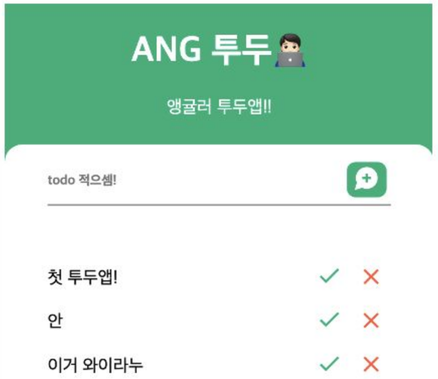
- 트랜스파일링 & 의존 모듈 번들링 (Angular CLI - **webpack** 사용해 하위 작업 자동화 지원)
  - **TypeScript**에서 **JavaScript**로의 트랜스파일링
  - 디버깅 용도의 **source map** 파일 생성
  - 의존 모듈과 **HTML, CSS,** 자바스크립트 번들링
  - AoT 컴파일
  - 코드의 문법 체크
  - 코드의 규약 준수 여부 체크
  - 불필요한 코드의 삭제 및 압축



“HELLO WORLD”

코드잼





Google Developer Student Clubs
Soongsil University

프론트엔드의 목적은 무엇일까요?

프론트엔드의 목적은 무엇일까요?

사용자에게 전달할 **데이터(모델)**를 **가공(서비스)**하는 수단

어떤 데이터를 다루어야 하는가

투두 데이터
- 내용
- 완료여부
- 고유 ID

STEP 1
서비스와 데이터

STEP 1
서비스와 데이터

어떤 데이터를 다루어야 하는가

투두 데이터
- 내용
- 완료여부
- 고유 ID

어떤 조작들이 필요한가

투두 추가
투두 내용 및 상태 조회
투두 상태 업데이트
투두 삭제

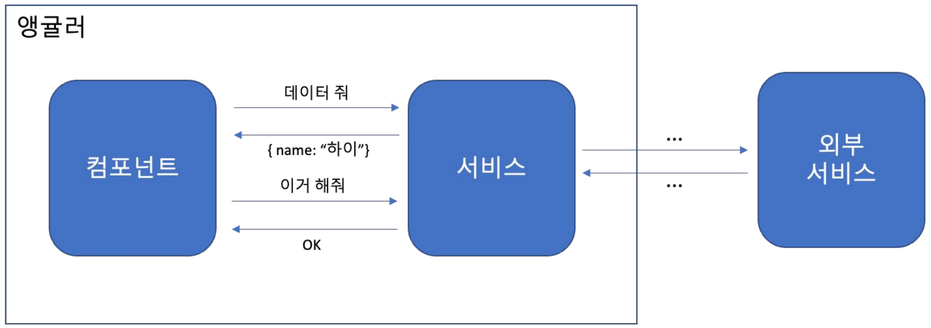
STEP 1
서비스와 데이터

어떤 데이터를 다루어야 하는가

투두 데이터
- 내용
- 완료여부
- 고유 ID

어떤 조작들이 필요한가

투두 추가
투두 내용 및 상태 조회
투두 상태 업데이트
투두 삭제



STEP 1
프로젝트 설계

어떤 데이터를 다루어야 하는가

투두 데이터
- 내용
- 완료여부
- 고유 ID

모델 클래스 구현

어떤 조작들이 필요한가

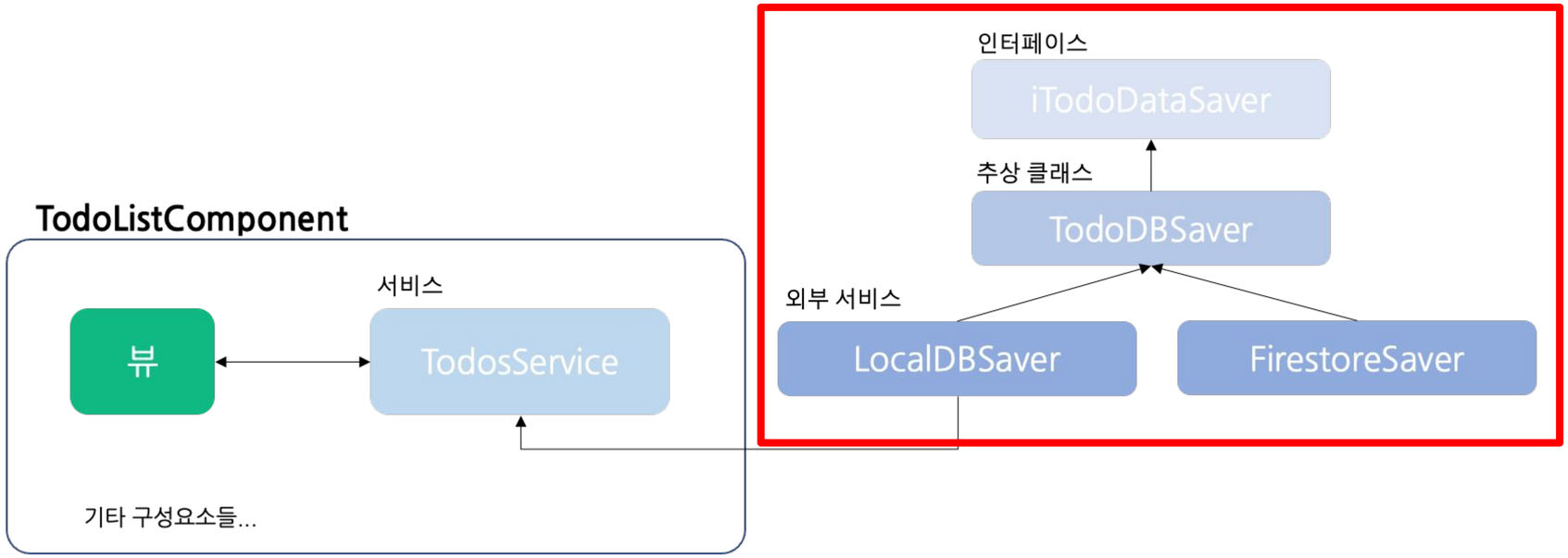
투두 추가
투두 내용 및 상태 조회
투두 상태 업데이트
투두 삭제

서비스 인터페이스 구현

확장성을 고려해보자..!

STEP 2

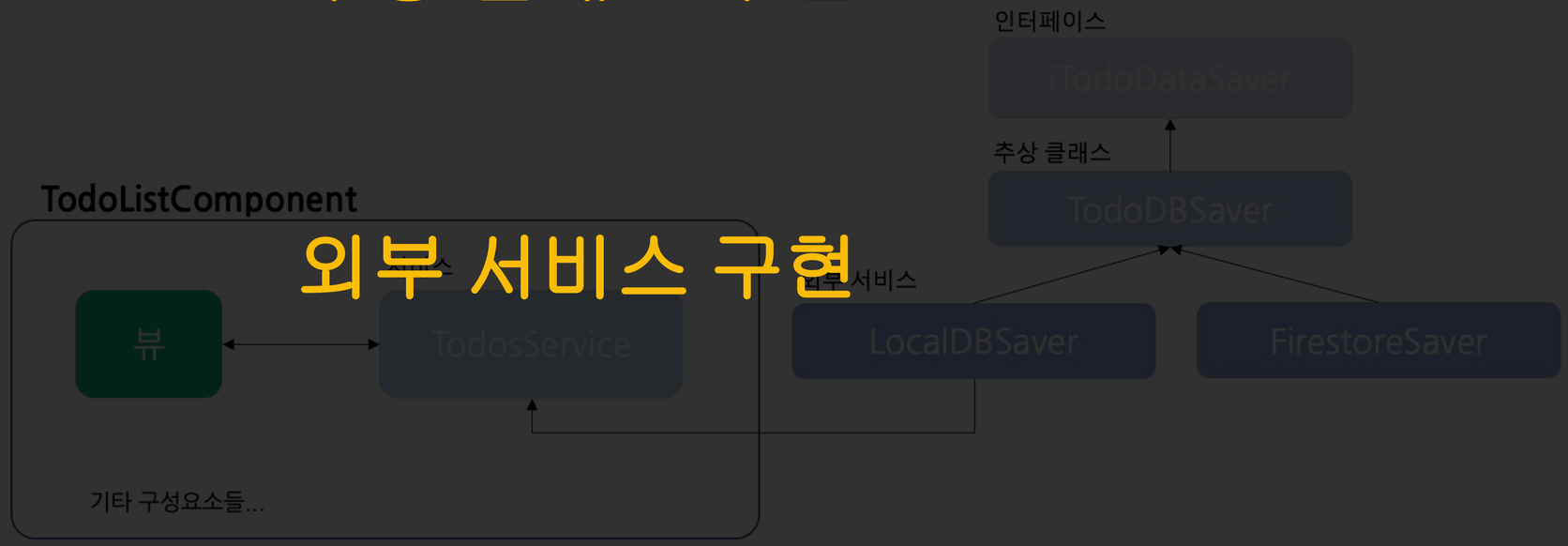
외부 서비스
설계 및 구현



확장성을 고려해보자..!

추상 클래스 구현

외부 서비스 구현



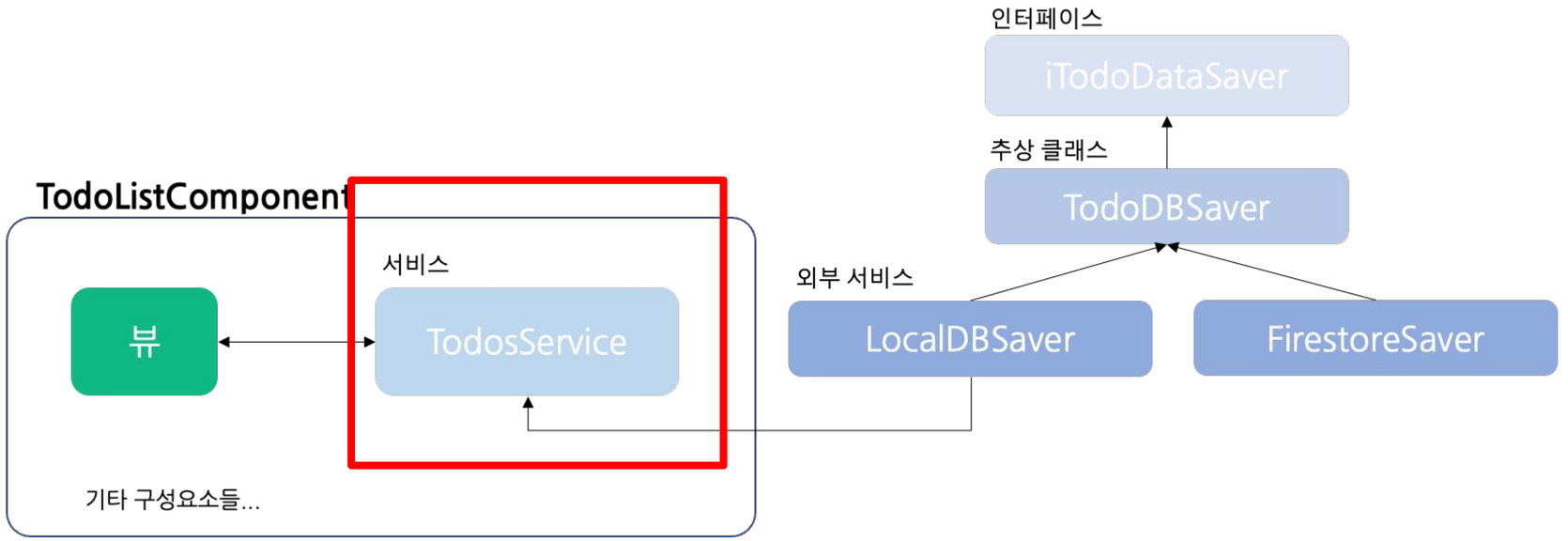
STEP 2

서비스
설계 및 구현

외부 서비스와 뷰 사이의 다리

STEP 2

내부 서비스 구현



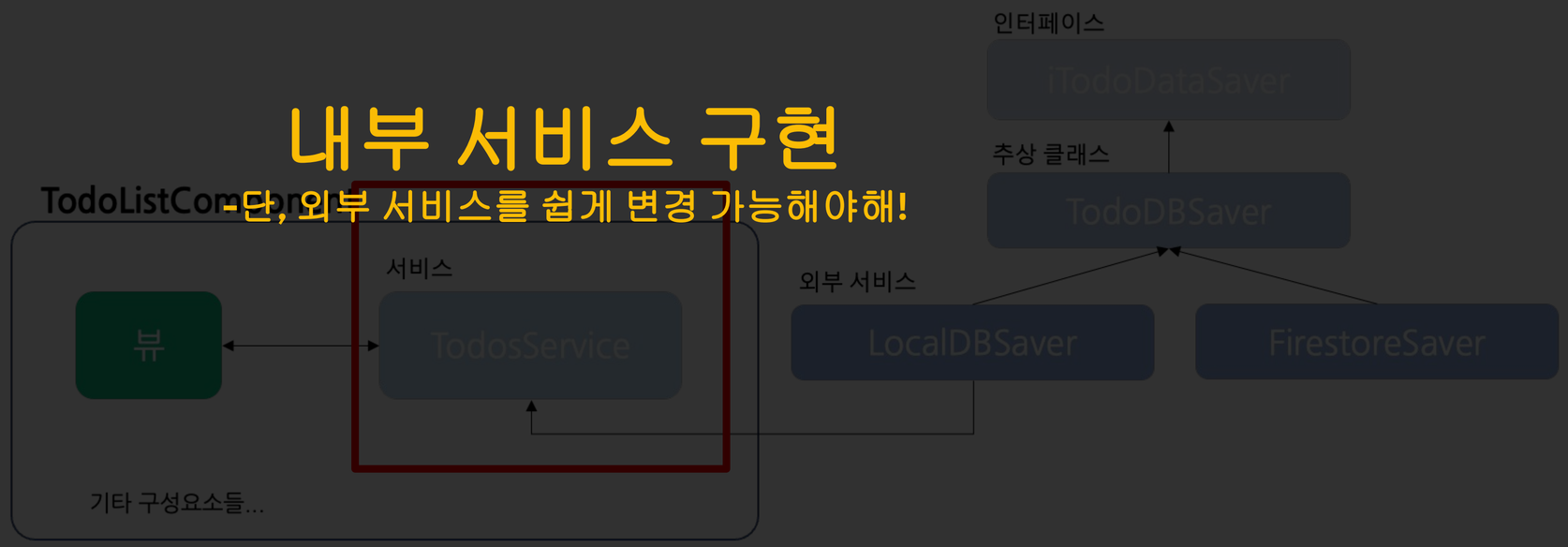
외부 서비스와 뷰 사이의 다리

STEP 2

내부 서비스 구현

내부 서비스 구현

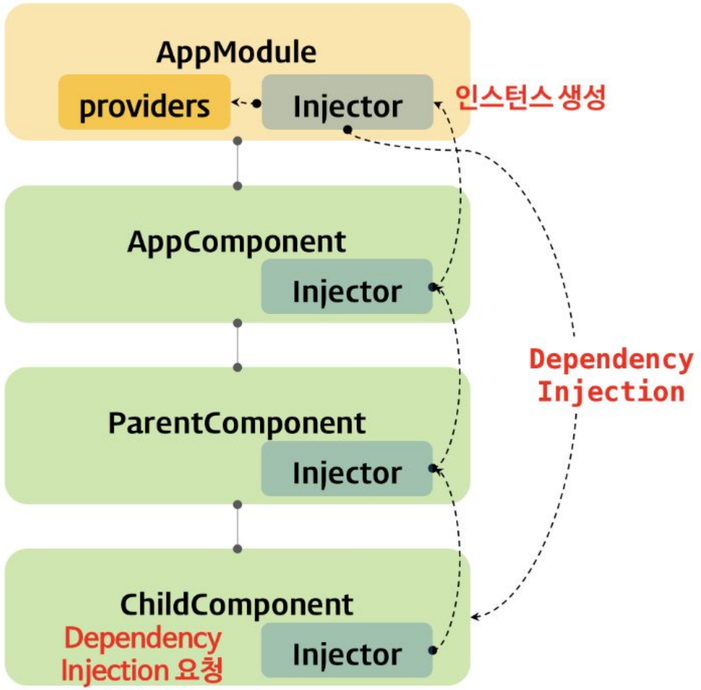
-단, 외부 서비스를 쉽게 변경 가능해야해!



데코레이터, @Injectable 그리고 모듈

STEP 2

내부 서비스 구현



인젝터 트리

STEP 3
컴포넌트 구현

사용자에게 데이터를 보여주자

헤더 (백그라운드?)

투두 컨테이너
- 느슨한 결합
- 디렉티브
- 더 좋은 구조

앵귤러 VS 리액트

+ 질문

마무리

감사합니다!

Google Developer Student Clubs
Soongsil University